Encrypting Matrix

Building a universal end-to-end encrypted communication ecosystem with Matrix and Olm

matthew@matrix.org
http://www.matrix.org

What is Matrix?

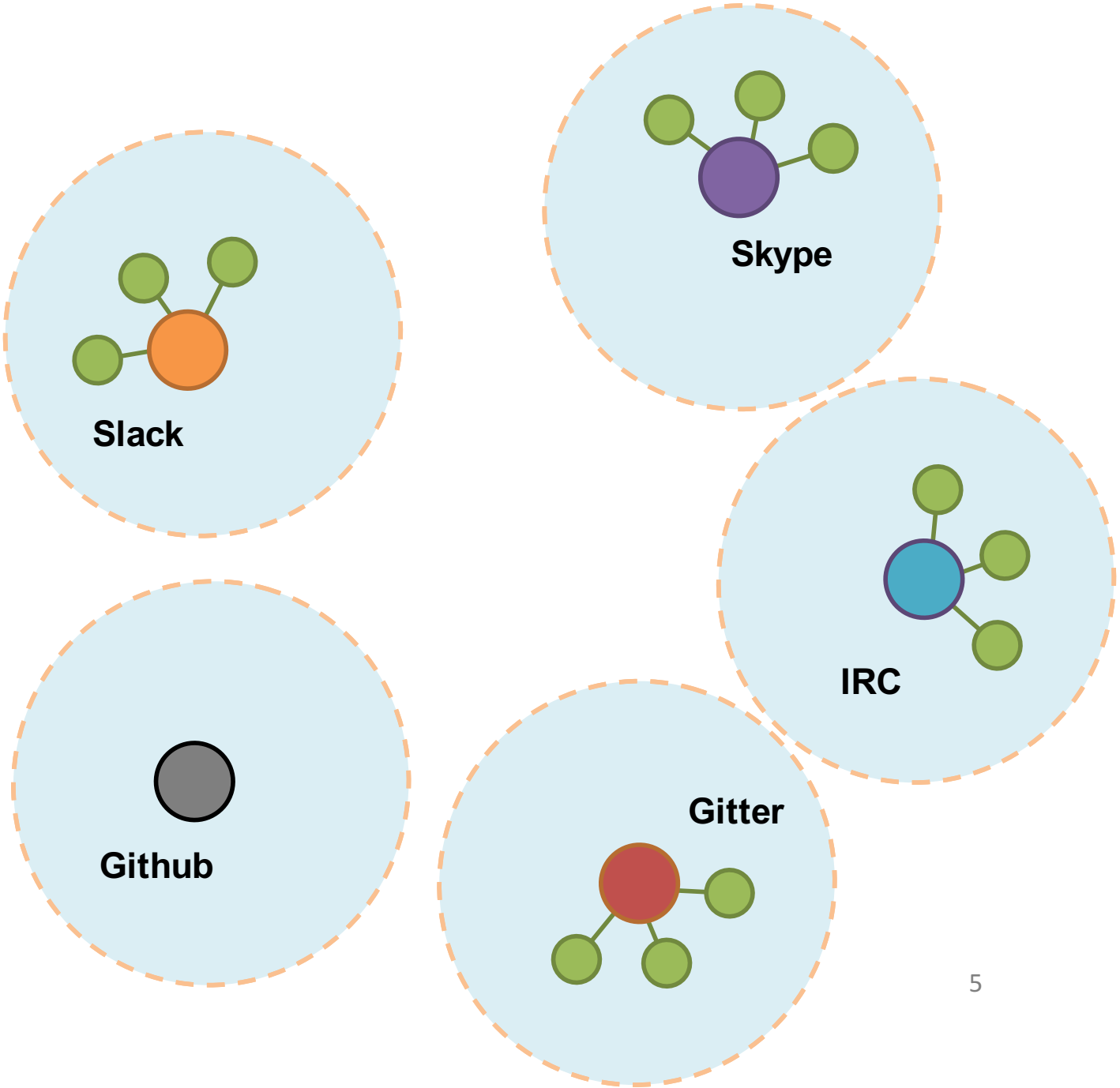
[matrix]

A non-profit open standard for defragmenting communication

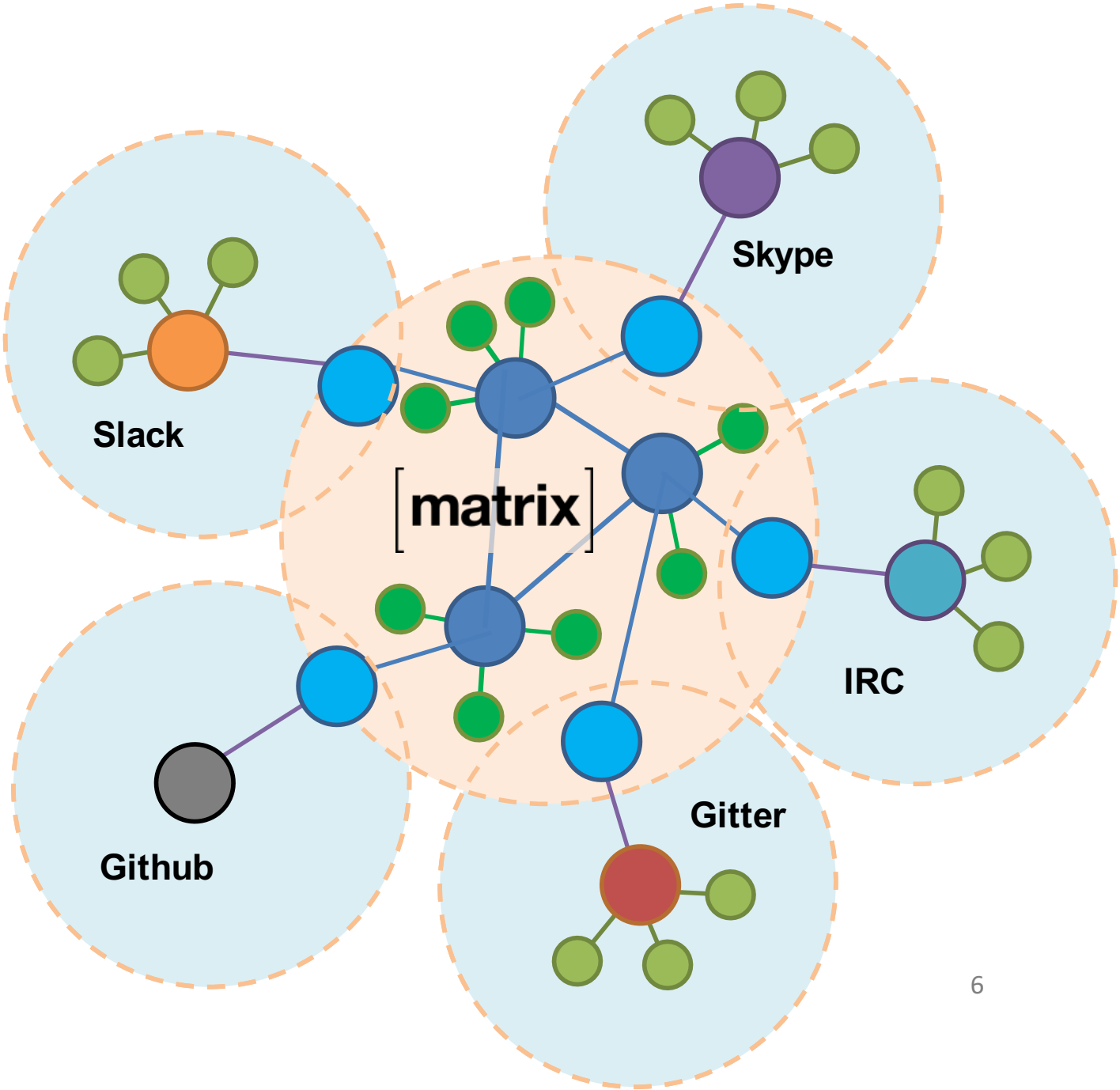
[matrix]

Creating a global encrypted communication meta-network that bridges all the existing silos & liberates our communication to be controlled only by us.

[matrix]



[matrix]



[matrix]

No single party owns your conversations.

Conversations are shared over all participants.

[matrix]

Use Matrix for:

Group Chat (and 1:1)
WebRTC Signalling
Bridging Comms Silos
Internet of Things Data

…and anything else which needs to pubsub persistent data to the world.


Why are you re-inventing XMPP!?!?

[matrix]

WE ARE NOT.

[matrix]

How is this different to XMPP?

- **Completely** different philosophy & architecture:
  - A single, monolithic, consistent, spec.
  - Different primitives:
    - Syncing decentralised conversation history (not message passing / pubsub)
    - Group conversation as a first class citizen
    - E2E crypto as a first class citizen
  - HTTP+JSON as the baseline API **(but you can use other transports too!)**
  - Core focus on defragmentation and bridging (hence the name “matrix”).

[matrix]

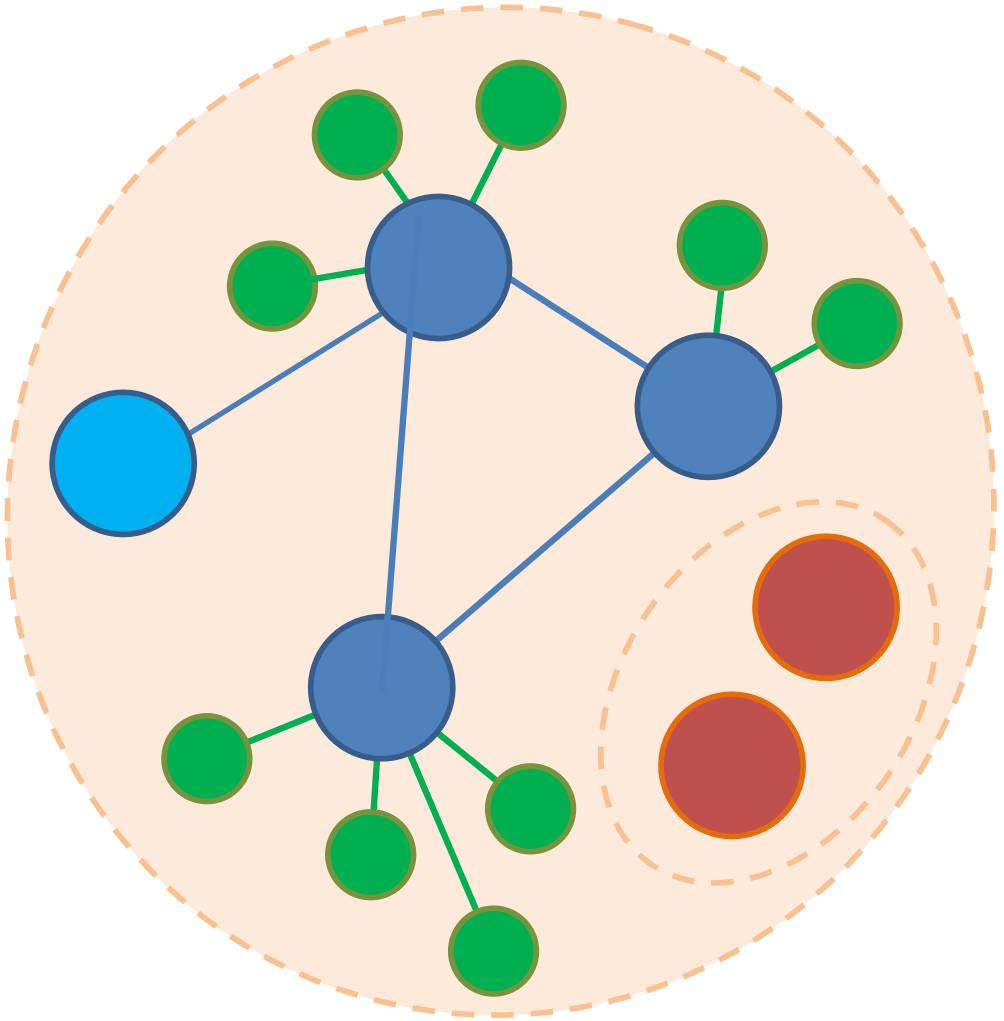
Matrix Architecture

Clients

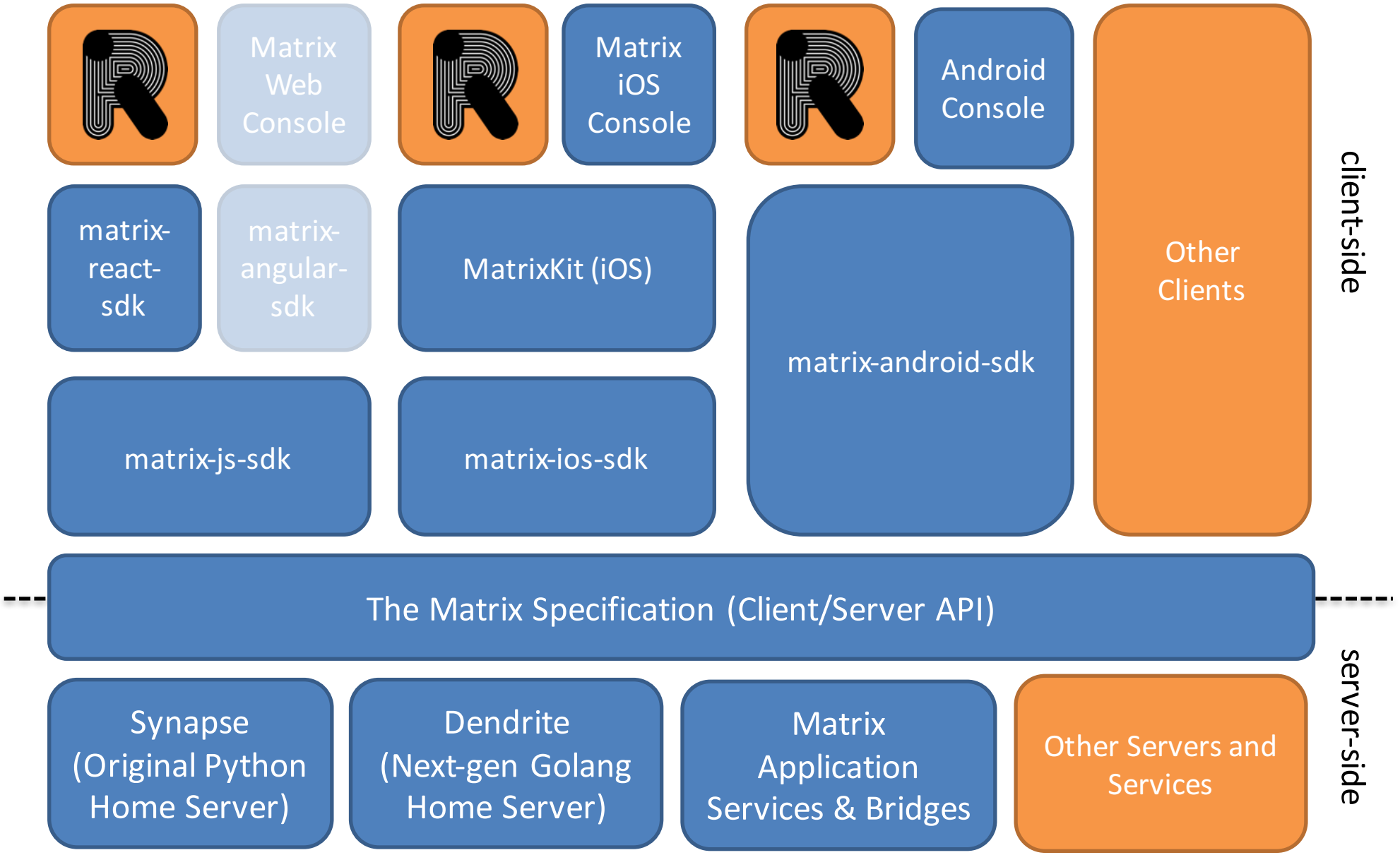
Home Servers

Application Servers

Identity Servers



The Matrix Ecosystem



[matrix]

What do you get in the spec?

- Decentralised conversation history (timeline and key-value stores)
- Group Messaging
- **End-to-end Encryption**
- VoIP signalling for WebRTC
- Server-side push notification rules
- Server-side search
- Read receipts, Typing Notifs, Presence
- Synchronised read state and unread counts
- Decentralised content repository
- “Account data” for users per room

How does it work?

https://matrix.org/#about

[matrix]

Clients

- >40 matrix clients (that we know about)
  - Ranging from text UIs (**Weechat**, Emacs(!))
  - …to desktop apps (Quaternion, **NaChat**, Pidgin)
  - …to glossy web and mobile clients (**Riot**)
  - …to protocol proxies (**matrix-ircd**)
- Over 15 client-side SDKs:
  - Official: JS, React, iOS, Android
  - Semi-official: Python, Perl5, Go
  - Community: Erlang, Ruby, Lisp, Elixir, Haskell, Rust…

[matrix]

Home servers

- **Synapse**: the original reference Matrix home server implementation from the core team.
  - 50K lines of Python/Twisted.
  - Some major perf and maintainability challenges…
- **Dendrite**: next-generation HS from the core team
  - ~10K lines of Golang
  - Work in progress, but alpha approaching soon…
  - Built around ”kafkaesque” append-only event logs
  - Scales horizontally.
- **Ruma**: Community project Rust implementation…
- BulletTime (Go), Pallium (Go), jSynapse (Java) experiments from the community

[matrix]

Latest Bridges!

- Official ones:
  - IRC
  - Slack
  - Gitter
  - Telegram
  - Rocket.Chat
  - MatterMost
  - FreeSWITCH
  - Asterisk (Respoke)
  - libpurple
- Community ones
  - Twitter
  - iMessage
  - Facebook Msgr
  - Hangouts
  - Slack webhooks
  - Gitter (‘sidecar’)
  - ~8 IRC ones…
  - ~4 XMPP ones...
  - ~3 Telegram ones…

What does it look like?

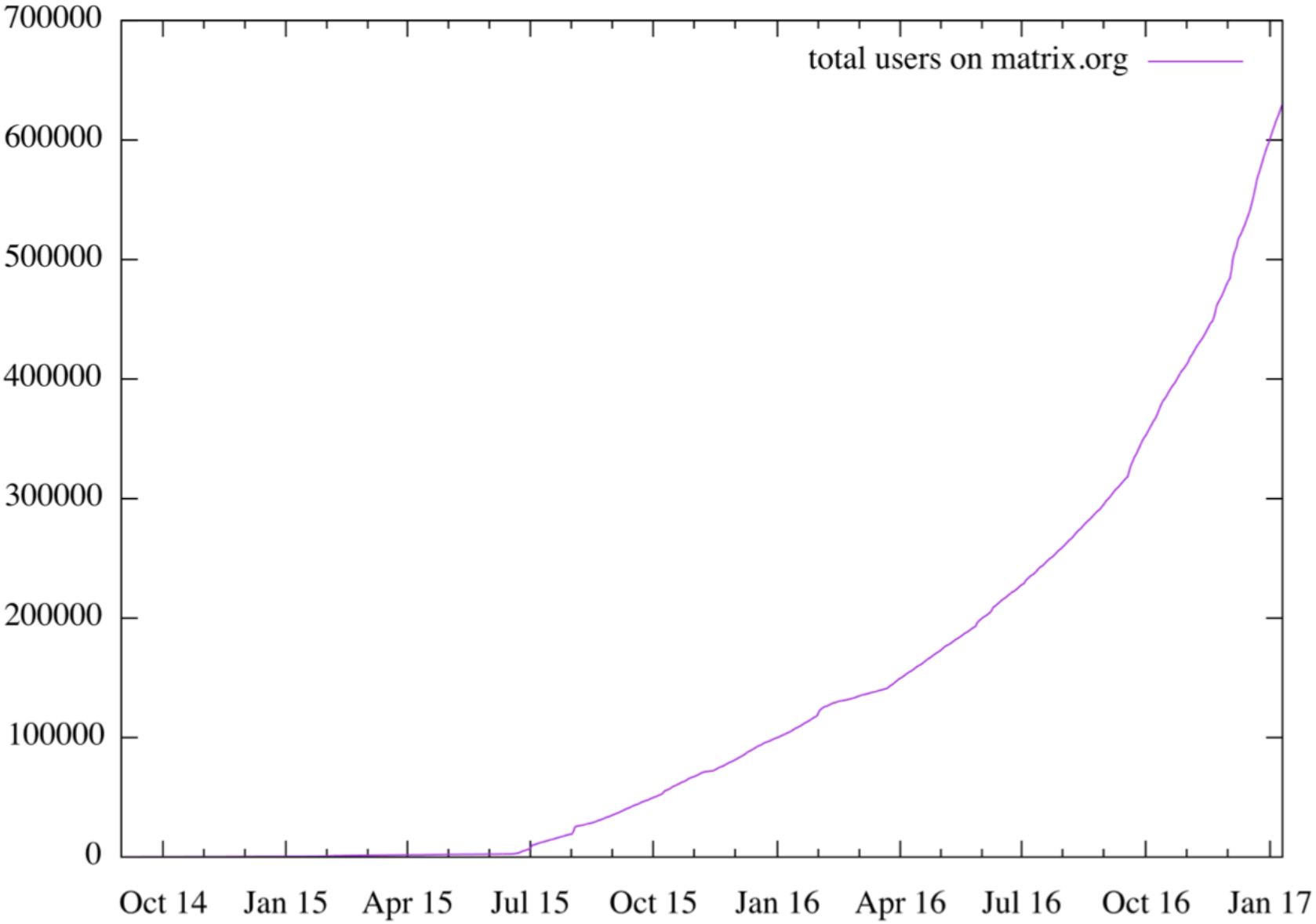
https://riot.im

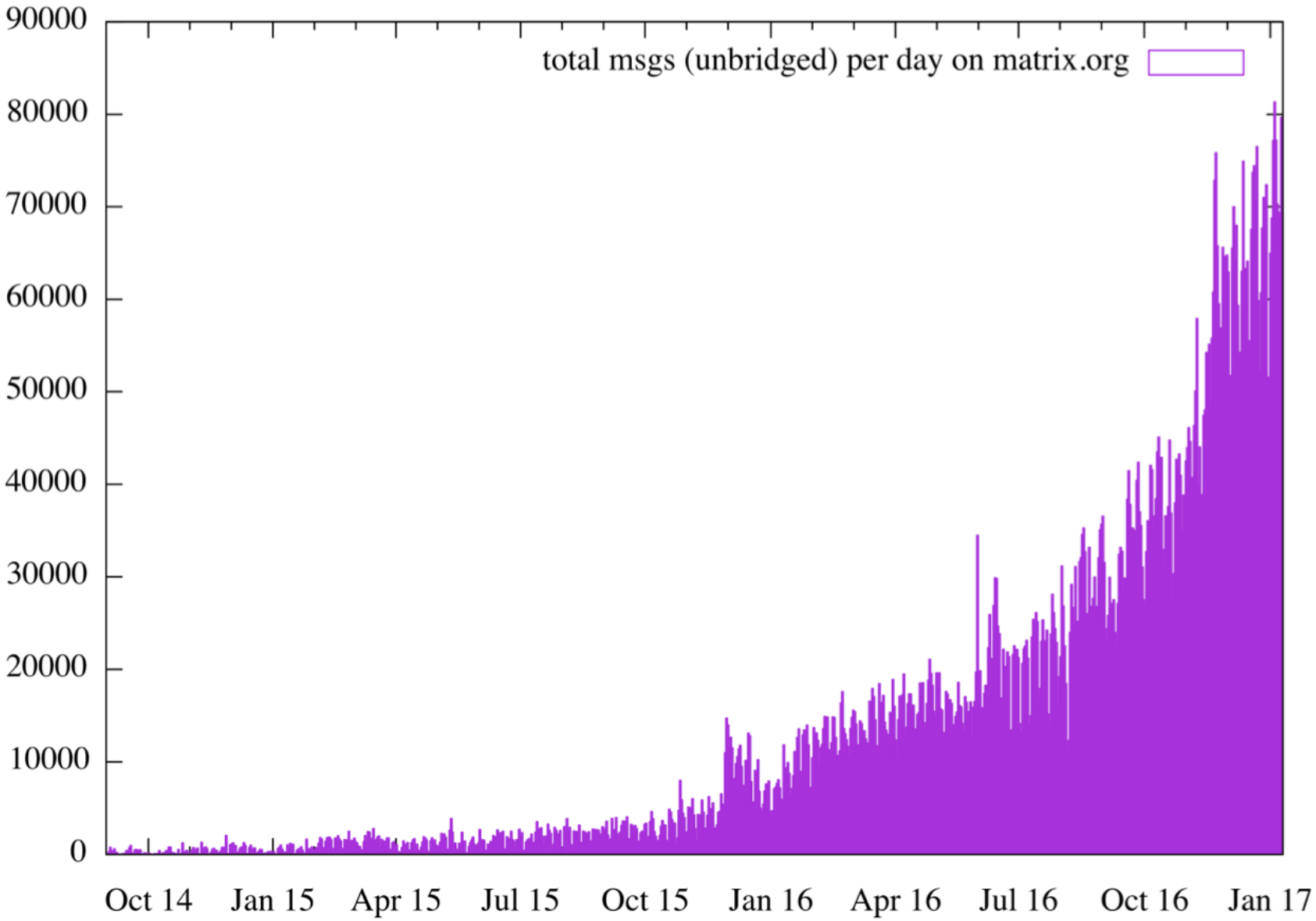
[matrix]

Community Status

- Started out in Sept 2014
- Currently in very late beta
- ~700K user accounts on the Matrix.org homeserver
- ~700K messages per day
- ~100K unbridged accounts
- ~100K unbridged messages per day
- ~70K rooms that Matrix.org participates in
- ~1500 federated servers
- ~1000 msgs/s out, ~10 msgs/s in on Matrix.org
- ~50 companies building on Matrix

[matrix]


total users on matrix.org
700000
600000
500000
400000
300000
200000
100000
0
Oct 14
Jan 15
Apr 15
Jul 15
Oct 15
Jan 16
Apr 16
Jul 16
Oct 16
Jan 17


total msgs (unbridged) per day on matrix.org
90000
80000
70000
60000
50000
40000
30000
20000
10000
0
Oct 14
Jan 15
Apr 15
Jul 15
Oct 15
Jan 16
Apr 16
Jul 16
Oct 16
Jan 17


End to End Crypto
with Olm
https://matrix.org/git/olm

[matrix]

Without end-to-end encryption, Matrix's replicated conversation history is a privacy problem.

[matrix]

➔ **Two years spent building decentralised E2E crypto into the heart of Matrix.**

[matrix]

Goals

- Configurable **trade-off between privacy and usability** per room.
  - Sometimes you want PFS…
  - …but sometimes you want to replay history.
- Encrypt & trust **per-device**, not per-user.
- Support **big rooms** (thousands of devices)
- Encrypt non-public rooms by default
- Be supported on all Matrix clients.

[matrix]

High level overview

- Two mechanisms at work:
  - **Olm** – a Double Ratchet implementation
    - provides a secure channel between two devices
    - used mainly for syncing key data
  - **Megolm** - a new ratchet that encrypts a sender's messages for a **group** of receivers
    - Ratchet state is shared to receivers 1:1 over Olm
    - Ratchets can be replaced to seal history
    - Ratchets can be fast-forwarded to share selective history

[matrix]

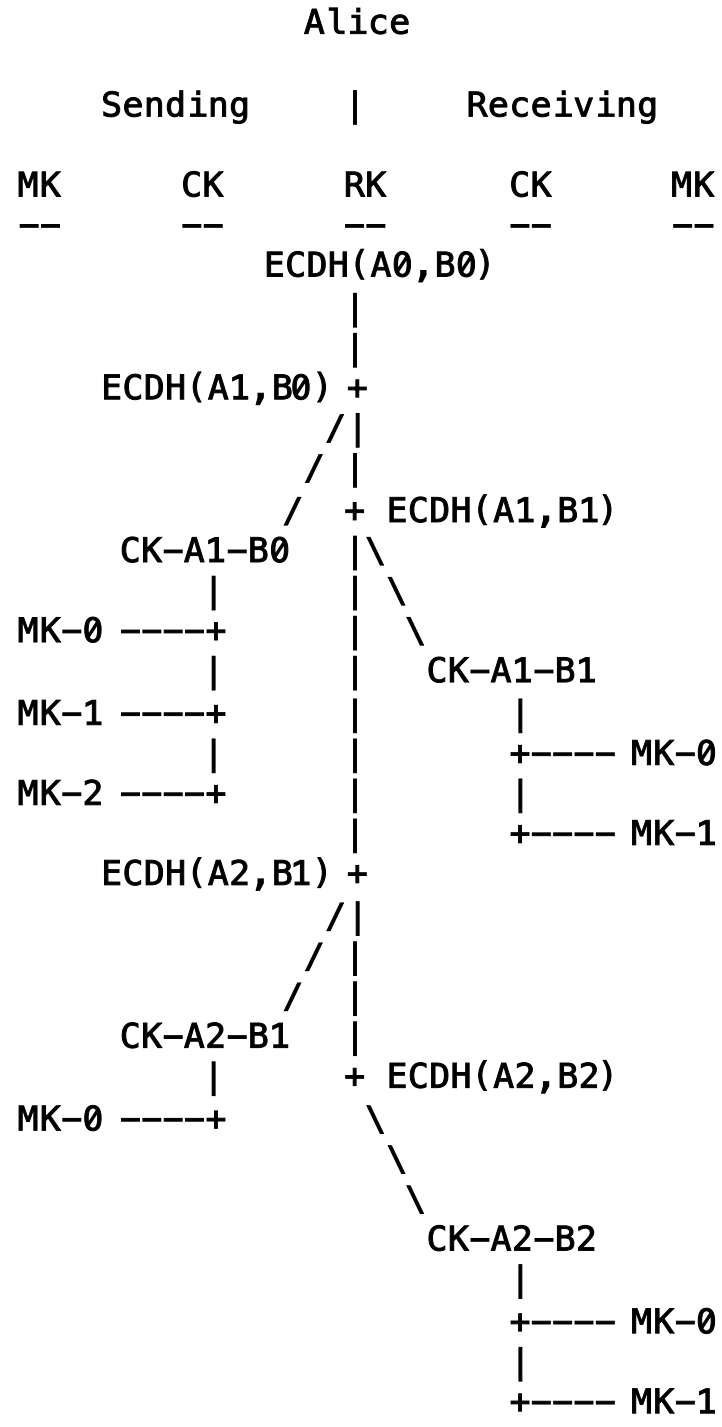
Key management

- Uses EC25519 keys.
- Keypairs generated **per-device** at login.
- Private keys are stored only on the device (duh).
- Public keys are published on your homeserver.
- Keys are verified by comparing public fingerprints.
  - **This is placeholder UX; we are looking at mnemonics, QR codes, cross-signing and other alternatives.**
- Attachments are AES-CTR encrypted (with integrity hash) using a new random key per file.

[matrix]

Olm

- New Apache licensed C++11 implementation of trevp/moxie's Double Ratchet Algorithm, exposing a C API: https://matrix.org/git/olm
- Formal spec: https://matrix.org/docs/spec/olm.html
- Supports encrypted async 1:1 communication.
- Chosen for quality & to avoid ruling out compat with WhatsApp etc.
- Defines a non-reversible series of keys for encrypting messages by advancing two ratchets; a hash ratchet and a ECDH ratchet.
- The ECDH ratchet advances when the message flow changes direction, spawning a new hash ratchet.
- Feb 2016: we encrypted each msg per recipient via Olm: $O(n^2)$. No way to share history.


Alice
Sending
Receiving
MK
CK
RK
CK
MK
ECDH(A0,B0)
ECDH(A1,B0) +
+ ECDH(A1,B1)
CK-A1-B0
MK-0
MK-1
MK-2
CK-A1-B1
MK-0
MK-1
ECDH(A2,B1) +
CK-A2-B1
+ ECDH(A2,B2)
MK-0
CK-A2-B2
MK-0
MK-1

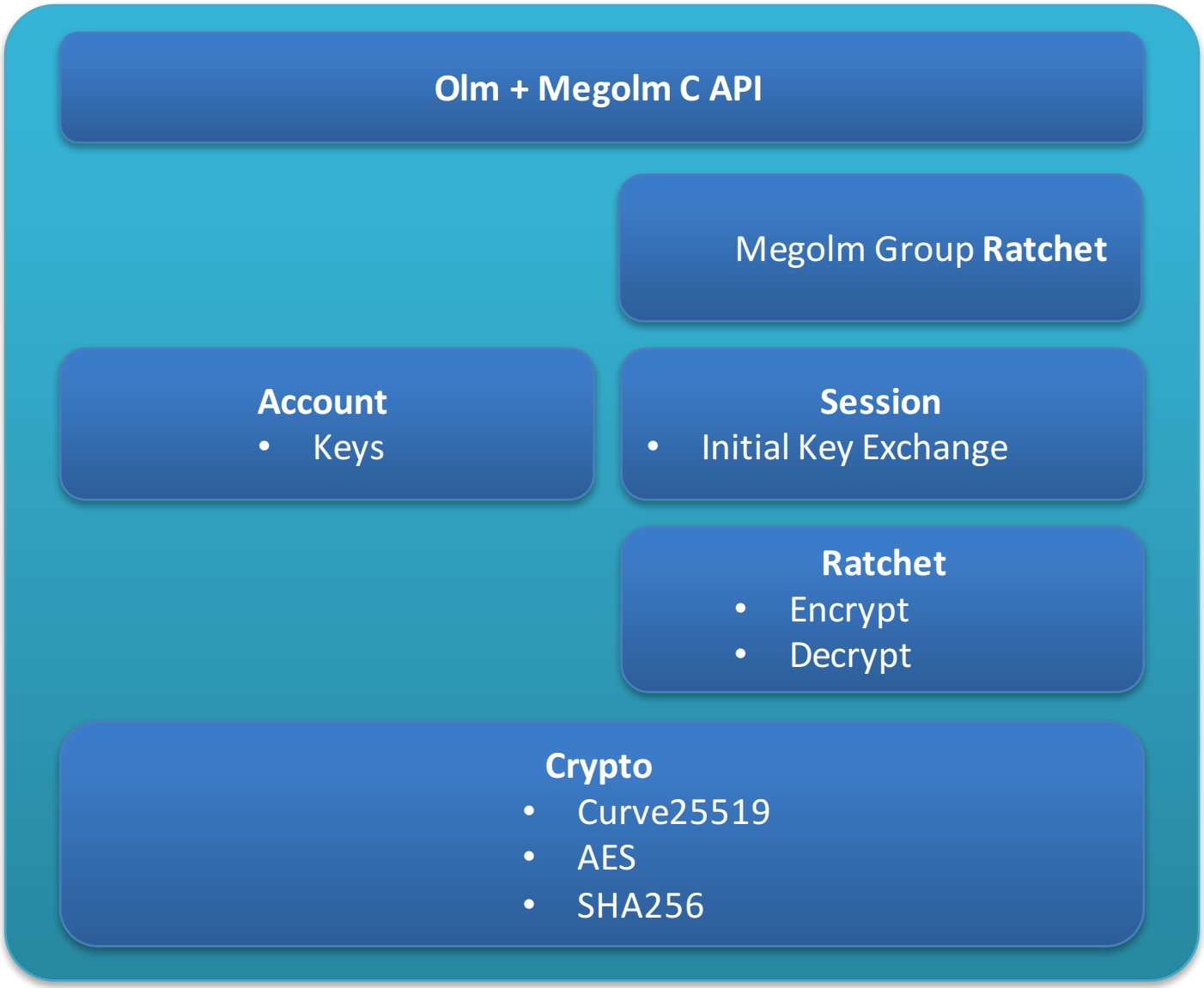
[matrix]

Megolm

- Entirely new ratchet for group chat with shareable history.
- Formal spec: https://matrix.org/docs/spec/megolm.html
- Each sender maintains a ratchet “aka outbound session” to encrypt messages they send to a room.
- The ratchet is shared with other participants via Olm (as “inbound sessions”). Uses new direct “to-device” messaging API in Matrix.
- Participants can save the ratchet key data to replay server history.
- The sender can choose to start a new ratchet at will, depending on the privacy desired – typically every N messages, or whenever a user leaves a room.
- An existing ratchet can be fast-forwarded before sharing, to lock the receiver out of being able to decrypt prior history.
- Nov 2016: Megolm beta starts

libolm

130KB of x86-64, 208KB of asm.js



[matrix]

Security Assessment

- libolm 1.3.0 assessed by NCC Group in Sept 2016
- Findings released to the public! https://www.nccgroup.trust/us/our-research/matrix-olm-cryptographic-review
- Olm: 2x low risk finding, 1x informational
- Megolm: 1x high, 1x medium, 4x low risk.
- 3 findings were features, not bugs (i.e. ability to configure a room for replaying history!)
- All findings fixed in libolm or the Matrix Client SDKs.
- No issues found in libolm since the audit!

Demo!

[matrix]

Architectural problems…

- Ironically, we may have **focused too much on libolm**.
- Reliably and efficiently **synchronising megolm ratchets** over a federated system like Matrix is **non-trivial**.
- More LOC than libolm itself, and in many ways more fiddly.
- You need to know precisely what devices are in a room when sending a message, so you can ensure your megolm ratchet is shared with them so they can decrypt your message…
- …so very prone to races, which we're still fixing currently.
- Heavily coupled to Matrix Client SDK for server interaction, so was implemented as part of the client SDKs…
- …resulting in 3 separate implementations (JS, ObjC, Java) of precisely the same logic. To be fixed in future?

[matrix]

Design problems…

- It's possible that Megolm is over-engineered.
- We can end up generating a **lot** of session keys, which must then be stored for decrypting history.
- Where do we put them all?
- Given we have so many sessions, why not share a new ratchet than fast-forward existing ones?
- **→ Plan is to see how well it works in practice & tune the session rate before rethinking.**

Goals checklist

- Configurable trade-off between privacy and usability per room.
  - **Supported in protocol** (but not really exposed yet in clients)
- Encrypt & trust per-device, not per-user.
  - **Done!**
- Support big rooms (thousands of devices)
  - **Done!**
- Encrypt non-public rooms by default
  - **Will be done once out of beta**
- Be supported on all Matrix clients.
  - **Not yet**. Considering a e2e proxy to ease migration, and/or providing a high level cross-platform helper library (which we really need whatever).

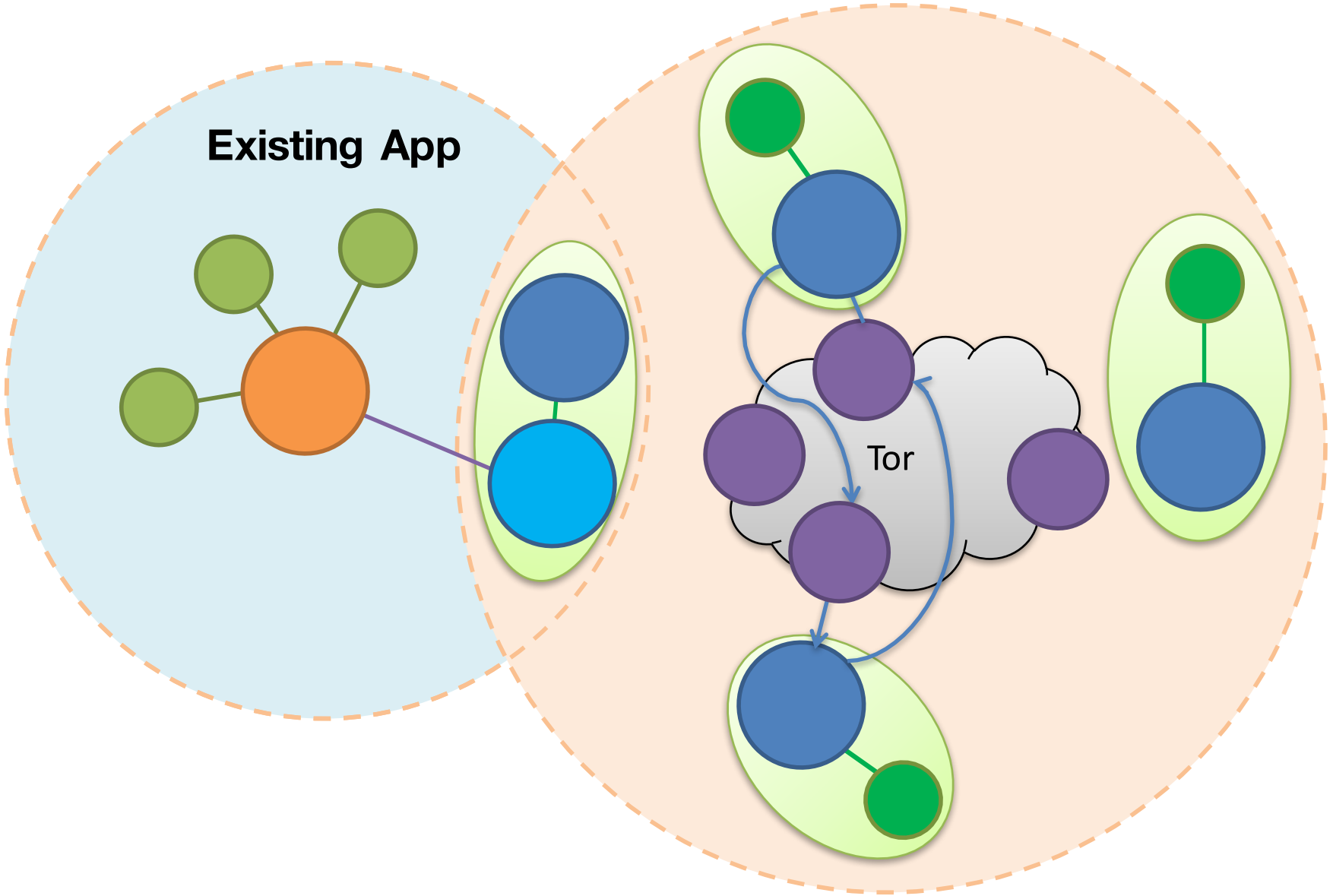
[matrix]

Metadata Privacy

- Matrix does not protect metadata currently; server admins can see who you talk to & when (but not what). If you need this today, look at Ricochet or Vuvuzela etc.
- Protecting metadata is incompatible with bridging.
- **However**, in future peer-to-peer homeservers could run clientside, tunnelling traffic over Tor and using anonymous store-and-forward servers (a la **Pond**).
- But for now this is sci-fi.

[matrix]

Matrix with Pond strategy



[matrix]

Latest release info

- Riot/Web 0.9.7 (released today!) gives:
  - Warning user properly on unknown devices
  - Ability to blacklist unverified devices by default
  - Backing up & restoring megolm session ratchet data
  - Entirely new device tracking API to improve session sharing reliability
  - “Rageshake” bug reporting to help debug when things fail
- Unfortunately E2E is definitely still in beta.
- Develop branches of Riot/iOS & Riot/Android are implementing the above too.

[matrix]

Olm: What's next?

- Ability to share session ratchet data with new devices or new room participants
- Cross-signing device keys?
- Better device verification
- Better push notification UX for E2E rooms
- Better primitives & performance
- Turning on E2E by default for rooms with private history
- Negotiating E2E with legacy clients(?)

[matrix]

Matrix: What's next?

- More hosted bridges, bots, services etc
- Threading
- Message tagging (e.g. "Like" support)
- Group ACLs
- File tagging and management
- Decentralised identity
- "Fixing spam"

We need help!!

[matrix]

- We need people to try running their own servers and join the federation.
- We need people to run gateways to their existing services
- We need feedback on the APIs.
- Consider native Matrix support for new apps
- Follow @matrixdotorg and spread the word!

Thank you!

matthew@matrix.org
http://matrix.org
@matrixdotorg

A Double ratchet.
Kinda sorta.

Alice

Bob

Alice and Bob both generate identity (I) & ephemeral (E) elliptic curve key pairs

Initial Shared Secret (ISS) =
ECDH(Ea, Ib) +
ECDH(Ia, Eb) +
ECDH(Ea, Eb)

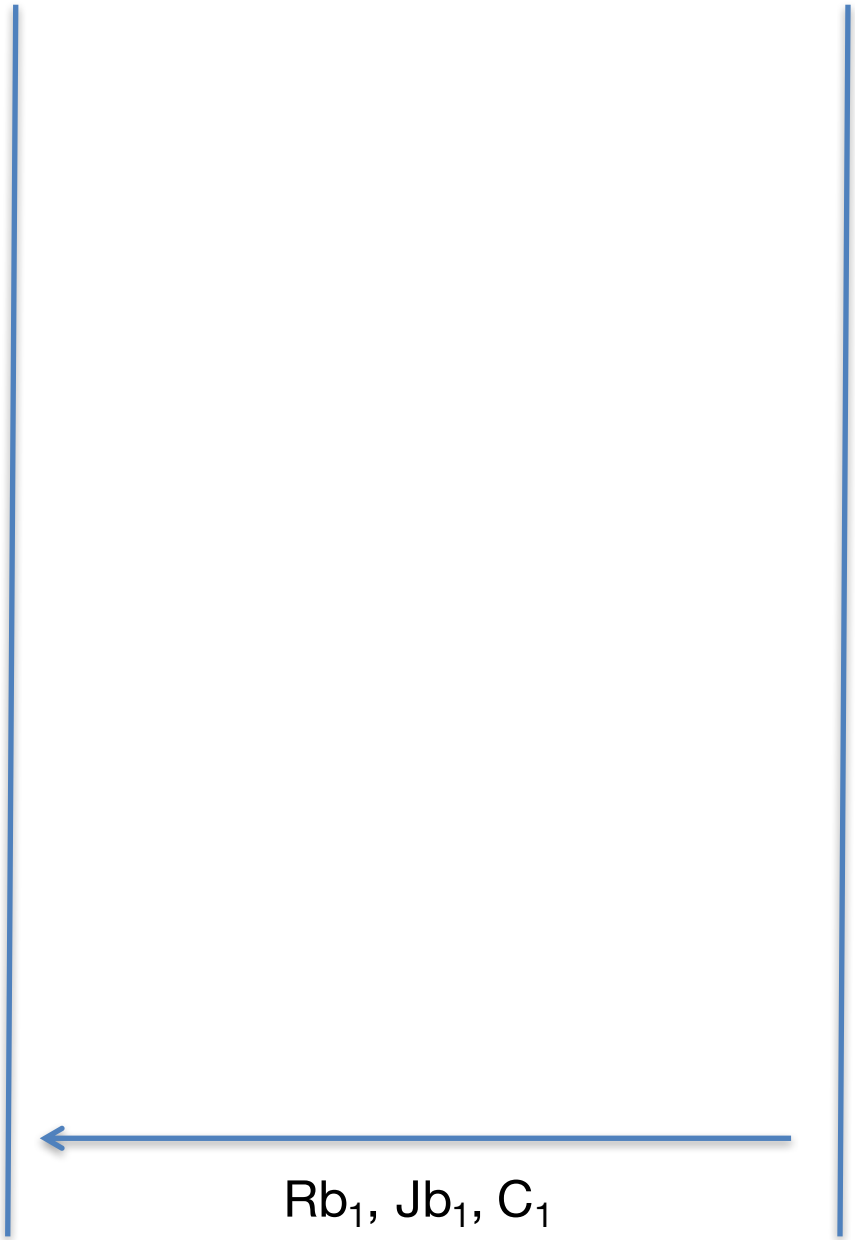
Discard Ea
Derive chain key from ISS (HMAC)
Derive message key (K_0) from chain key (HMAC)
Derive new chain key ← **hash ratchet**
M_0 = Message plaintext
C_0 = Authenticated Encryption of (M_0, K_0)
Ra_0 = generate random ratchet key pair
Ja_0 = incremental counter for each hash ratchet advancement

Ia, Ea, Eb, Ra_0, Ja_0, C_0

A Double ratchet.
Kinda sorta.

Alice

Bob



Compute same Initial Shared Secret =
ECDH(Ea, Ib) +
ECDH(Ia, Eb) +
ECDH(Ea, Eb)

Compute same K_0
M_0 = Authenticated decryption of (C_0, K_0)

To respond, B starts new ratchet chain:
Rb_1 = generate random ratchet key pair
New Initial Shared Secret =
ECDH(Ra_0, Rb_1) ← **ECDH Ratchet**

C_0 = Authenticated Encryption of (M, K_0)
Ra_0 = generate random ratchet key
Ja_0 = incremental counter for each hash ratchet advancement

The client-server API

To send a message:

```
curl -XPOST -d '{"msgtype":"m.text", "body":"hello"}'
"https://alice.com:8448/_matrix/client/api/v1/rooms/ROOM_
ID/send/m.room.message?access_token=ACCESS_TOKEN"
```

```
{
    "event_id": "YUwRidLecu"
}
```

The client-server API

To set up a WebRTC call:

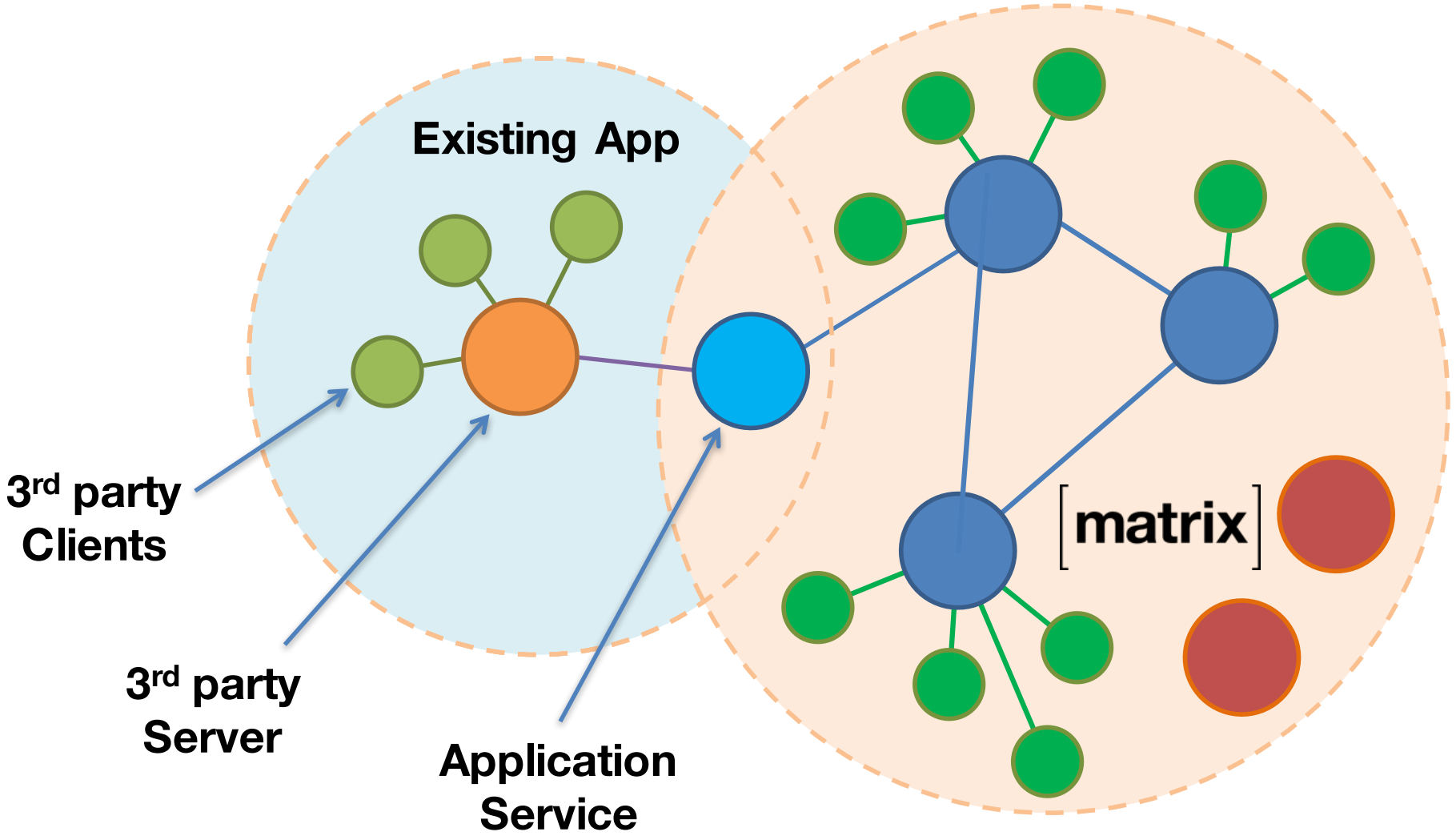
```
curl -XPOST -d '{\
  "version": 0, \
  "call_id": "12345”, \
  "offer": {
    "type" : "offer”,
    "sdp" : "v=0\r\no=- 658458 2 IN IP4 127.0.0.1…"
  }
}'
"https://alice.com:8448/_matrix/client/api/v1/rooms/ROOM_
ID/send/m.call.invite?access_token=ACCESS_TOKEN"

{ "event_id": "ZruiCZBu” }
```

Basic 1:1 VoIP Matrix Signalling

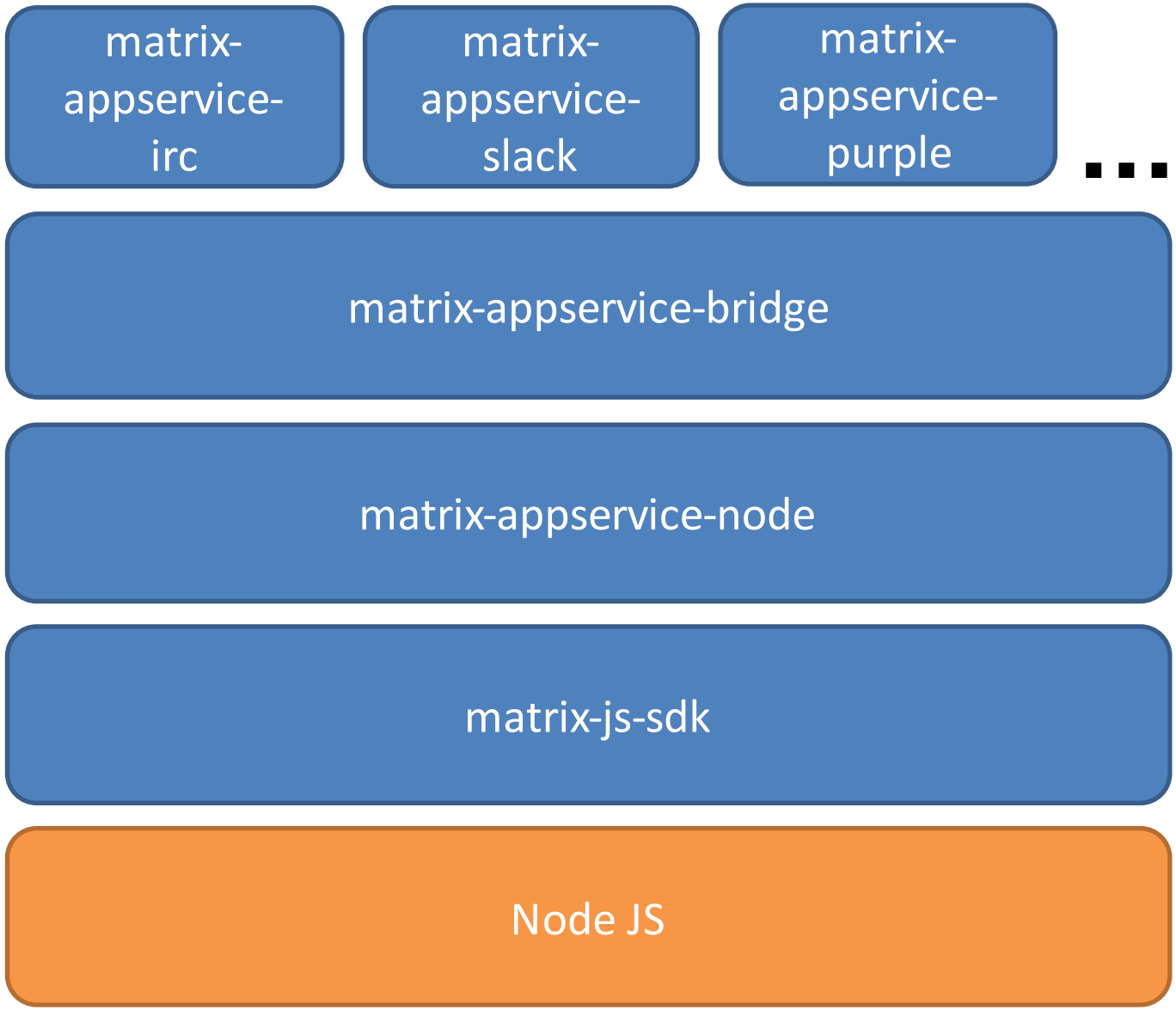
```
    Caller                          Callee
m.call.invite ----------->
m.call.candidate -------->
[more candidates events]
                              User answers call
                   <------ m.call.answer
                [media flows]
                   <------ m.call.hangup
```

[matrix]

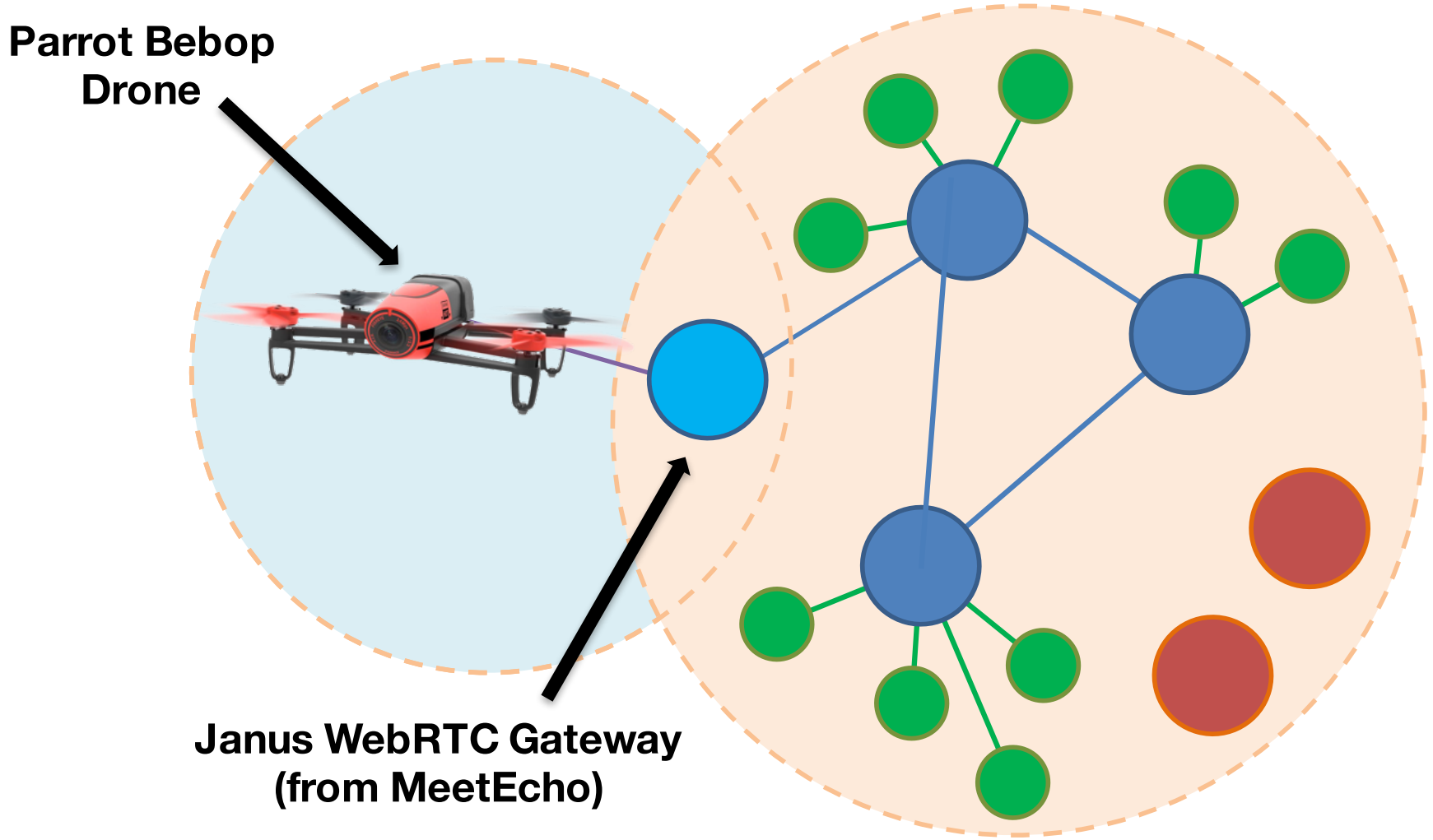
Bridges and Integrations



Typical Bridging Stack



Matrix to IOT…



https://www.youtube.com/watch?v=D7jZSYkXqt4&t=2649

[matrix]

Matrix and VR...